Licenciatura em Engenharia
Informática e Computadores

- Segurança Informática-

19/20 SI

2ª Série de Exercícios -> Resolução Grupo 6 LI51N

Docente: José Simão

Alunos: Luís Guerra nº 43755
David Albuquerque nº43566

Hugo Almeida nº 42931

1)

1.1)

Durante a parte de autenticação do processo handshake, o cliente realiza várias verificações criptográficas para assegurar que o certificado enviado pelo servidor é autêntico

A assinatura digital é utilizada no contexto de autenticação do cliente, em que o cliente pede as raízes de confiança do servidor, se o certificado for compatível com outos CA'S envia. O servidor valida o certificado do cliente usando os CA's do servidor. Depois, o cliente assina as mensagens anteriores e envia para o servidor a assinatura. Depois é validada a assinatura com a chave pública.

Teoricamente, poderíamos usar o MAC em vez da assinatura digital, mas como o MAC é simétrico, o cliente e o servidor teriam de usar a mesma chave. O handshake TLS não garante isso logo para usar o esquema MAC teríamos de fazer isso antes.

1.2)

Nesta parte do livro apresentado é posto em causa o uso seguro do protocolo TLS, que é apresentado como "perfect forward secrecy" ou "PFS". O que esta técnica pretende é que haja um tipo de defesa contra o atacante que guarda comunicações encriptadas visto que as "chaves de sessão" são encriptadas só com as chaves de longo prazo dos 2 lados.

Basta o atacante obter estas as chaves acima referidas e o processo de decriptação é muito fácil podendo ser descoberta e adquirida toda a comunicação pelo atacante.

Estas chaves podem ser obtidas de várias maneiras:

- Caso o cliente ou o servidor sejam atacados por outra entidade e a chave privada for recuperada.

- Uma chave de longo prazo recuperada de um dispositivo que foi vendido ou desativado sem ter sido previamente apagada.

- Uma chave de longo prazo usada num dispositivo como uma chave predefinida ou "default".

- Uma chave gerada por uma "third-party" confiável como por exemplo um certificado, e posteriormente é recuperada através de extorsão ou até compromisso.

- Através de um improvável avanço de técnicas criptográficas, ou o uso de chaves assimétricas com tamanho insuficiente.

- Ataques do tipo sociais (Engenharia social) contra administradores de um sistema, comprometendo-o.

- Através de um registo de chaves privadas que foi protegido inadequadamente.

O PFS contorna alguns deste tipo de ataques tornando infazível para um atacante descobrir as chaves de sessão mesmo que as chaves de sessão já tivessem sido comprometidas.

2)

No RFC 8018 o salt tem o papel de produzir um grande “set” de chaves correspondentes a uma única password, uma das quais é escolhida aleatoriamente de acordo com o salt. Uma chave desse “set” é escolhida através da aplicação de uma função de derivação de uma chave, do tipo: DK = KDF (P, S). DK= chave derivada, P = password, S = salt.

Os benefícios que este processo traz são os seguintes:

- É difícil para um atacante pré-computorizar todas as chaves, ou até as chaves mais prováveis que correspondem a um dicionário de passwords. Se o salt for de 64 bits, por exemplo, terá no máximo 2^64 chaves para cada password o que torna infazível para o atacante determinar.

- É extremamente improvável que a mesma chave seja escolhida duas vezes seguidas. Retomando o exemplo acima referido, se o salt tiver 64 bits, a chance de colisão de 2 chaves não é significativa até que 2^32 chaves tenham sido produzidas. O facto de que as colisões são improváveis causa algumas preocupações sobre as interações entre vários usos da mesma chave que pode suscitar quando estamos a usar algumas técnicas de encriptação e autenticação.

3)

Quando estamos num browser e fazemos um pedido de login para a aplicação Web (user = a , password = x) ela recorre à sua base de dados de verificação de informação válida para ver se esse utilizador realmente existe, caso exista, envia uma resposta de 200 OK para o browser e de seguida o browser comunica com a sua base de dados de cookies e realiza a operação set_cookie: id = x(x||T(a,k) onde T é um gerador de marcas e k é a chave gerada pela aplicação web. A instrução set_cookie é um header de resposta. Esta altura é altura de geração de cookie. Após esta etapa o browser faz um pedido de GET para a aplicação web em que faz a instrução cookie: id = x(x||T(a,k) em que cookie é um header de pedido e se bater certo aplicação autentica a cookie através uma resposta de 200 OK, sendo esta etapa a etapa de autenticação.

4)

4.1)

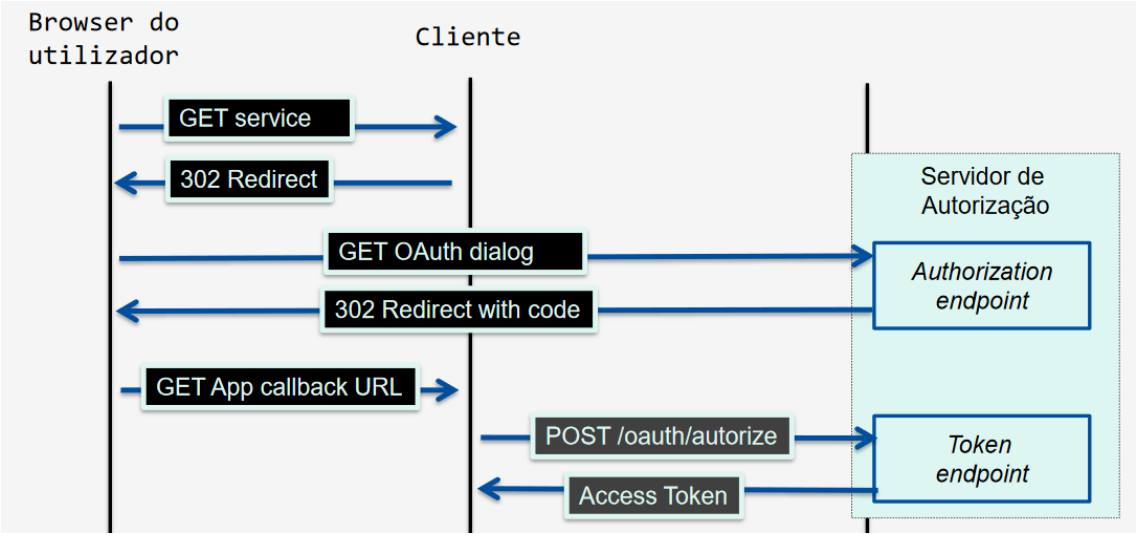
O objetivo do parâmetro scope é para limitar o acesso de uma aplicação a um token de acesso, ou seja, representam o tipo de autorização que está a ser pedido a um determinado recurso. Uma aplicação pode fazer o pedido de vários tokens mas a limitação exercida pelas scopes oferece uma proteção de dados ao utilizador. Scope é do tipo string em que cada pedido pode ter 0 ou mais scopes. O protocolo OAuth não define qualquer valor dos scopes visto que é altamente dependente da arquitetura interna do meio onde está a ser aplicada.

4.2)

O client_secret não é possível ser visto pelo atacante pois o client_secret é apenas trocado entre a aplicação web e o Identity Provider, ou seja, nunca passa pelo “front channel”. O client_id é possível que o atacante saiba, pois, antes de ser feito o processo de autenticação, é feito um pedido de autenticação com o client_id pelo browser e como o atacante tem acesso ao browser o cliente_id ficará comprometido.

4.3)

Depois de o servidor de autorização autorizar e fazer um 302 Redirect com o code, o browser faz um GET á aplicação para fazer um callback do URL, e a aplicação realiza um POST com oauth autorize e o servidor (no Token endpoint) retorna um access token para aplicação. O access token contém apenas informação de autorização, ou seja, não contem a identidade do utilizador logo essa informação não é passada á aplicação web.



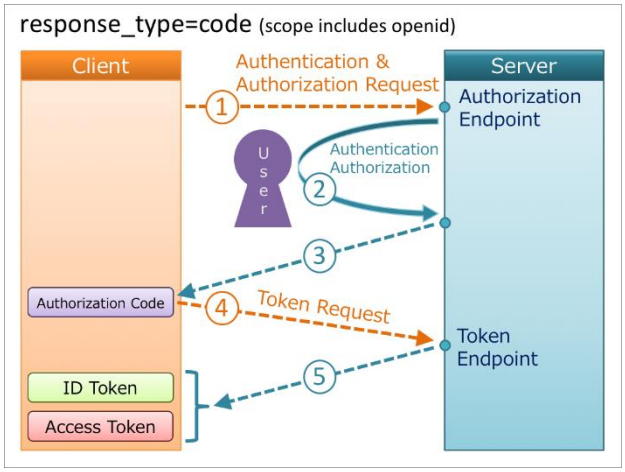
5)

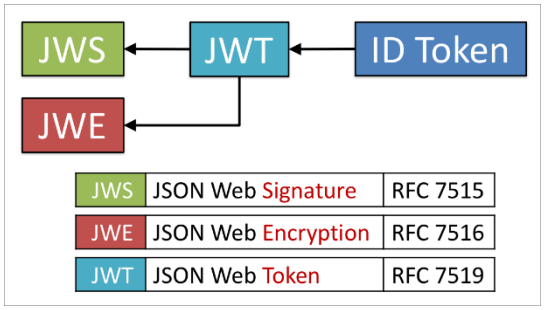
O cerne do protocolo OpenIDConnect é baseado no conceito de ID Tokens, que é um novo tipo de token que o servidor de autorização irá retornar que codifica a informação de autenticação do utilizador. Os ID Tokens são para ser entendidos pela aplicação “third party”, ao contrário dos access tokens.

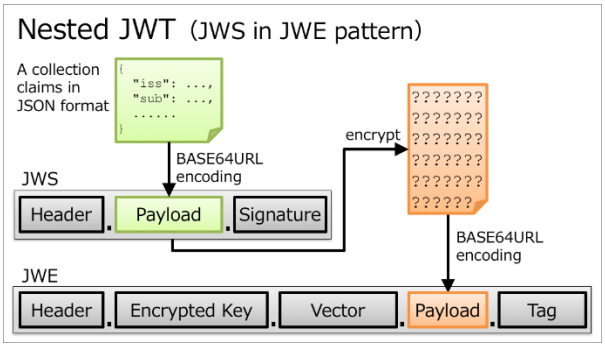
O OpenIDConnect acrescenta uma camada de identidade ao OAuth 2.0.

Os ID Tokens são uma estrutura JSON, mais especificamente uma JWT (JSON Web Token), que é uma parcela JSON assinada com a chave privada do emissor e que pode ser verificada pela aplicação web.

Dentro do JWT o ID Token contém um conjunto de claims (asserções) sobre o end-user no que é gerado pelo Identity Provider e enviado para p uma aplicação web registada com o Auth, que depois envia uma cookie com a autenticação ao browser.







6)

Preparação do exercício 6 a):

Gerar pems:

openssl x509 -inform der -in secure-server.cer -out secure-server-cer.pem

openssl x509 -inform der -in CA1.cer -out secure-server-CA1-cer.pem

openssl pkcs12 -in secure-server.pfx -out secure-server-key.pem –nodes

Editar “C:\Windows\System32\drivers\etc\hosts” e adicionar:

127.0.0.1 www.secure-server.edu

Instalar CA1.cer em “Place all certificates in the following store > Trusted Root Certification authorities”

Instalar CA1-int.cer automaticamente

Instalar Alice_1.pfx no chrome “Settings > Advanced > Privacy & Security > Manage Certificates > Personal > Import”